

Figure 1

FIGURE 2a

```

////////////////////////////////////
// Declaration of a pipelined 16 x 16 //
// unsigned multiplier                //
////////////////////////////////////

RESOURCEDEF MULT16x16_FULLPIPE_UNSIGNED
{
    //////////////////////////////////
    // A Multiplier                //
    //////////////////////////////////
    FUNCTIONALITY MULT;

    //////////////////////////////////
    // The instantiation code for a  //
    // specific multiplier           //
    //////////////////////////////////
    ATTRIBUTE INSTANTIATION
    {
        //////////////////////////////////
        // component_name is the specific soft IP      //
        // instance that needs to be accessed          //
        //////////////////////////////////

        attribute +
            "input wrap unsigned fixed[16,0] "+ component_name + "_A;\n" +
            "input wrap unsigned fixed[16,0] "+ component_name + "_B;\n" +
            "output wrap unsigned fixed[32,0] "+ component_name + "_R;\n";

        attribute +

```

```

        "instantiate mult16x16_fullpipe_unsigned : " + component_name +
        "(" +      "A = " + component_name + "_A," +
        "B = " + component_name + "_B," +
        "clk = " + clock_name + "," +
        "clr = " + reset_name + "," +
        "R = " + component_name + "_R" +
        ");";
    }

```

```

////////////////////////////////////

```

```

// Whether the Soft IP core can //

```

```

// perform the multiplication //

```

```

////////////////////////////////////

```

```

ATTRIBUTE CAN_DO

```

```

{

```

```

    <MULT>

```

```

    {

```

```

        if(in1->bitwidth() < 17 && in2->bitwidth() < 17 &&

```

```

            in1->is_unsigned() == true && in2->is_unsigned() == true)

```

```

        {

```

```

            attribute + "true";

```

```

        }

```

```

    else

```

```

    {

```

```

        attribute + "false";

```

```

    }

```

```

}

```

```

}

```

Figure 2b

////////////////////////////////////

// The Pipeline latency. I.e. the //  
// number of clock cycles after //  
// which new data can be fed to the //  
// pipelined multiplier //  
////////////////////////////////////

ATTRIBUTE PIPE\_DELAY

```
{  
  <MULT>  
  {  
    attribute + "1";  
  }  
}
```

////////////////////////////////////

// Is this a Combinatorial multiplier //  
// or a Sequential multiplier. This //  
// decides if this multiplier can be //  
// chained or not //  
////////////////////////////////////

ATTRIBUTE COMBINATIONAL

```
{  
  <MULT>  
  {  
    attribute + "false";  
  }  
}
```

figure 2c

```

}

////////////////////////////////
// The multiplier latency. I.e. the //
// number of clock cycles after    //
// which processing is over        //
////////////////////////////////

```

#### ATTRIBUTE NUM\_STATES

```

{
    <MULT>
    {
        attribute + "6";
    }
}

```

```

////////////////////////////////
// Interface access mechanism wherein //
// we have fixed latency of 6 clock //
// cycles with a throughput of 1    //
////////////////////////////////

```

#### ATTRIBUTE INTERFACE

```

{
    <MULT>
    {
        attribute + "state1: {";
        attribute + component_name + "_A = " + in1->name() + " ; \n";
        attribute + component_name + "_B = " + in2->name() + " ; \n";
    }
}

```

Fig 2a

```

attribute + "goto state2 ;\n";
attribute + "}";
attribute + "state2: {";
attribute + "goto state3 ;\n";
attribute + "}";
attribute + "state3: {";
attribute + "goto state4 ;\n";
attribute + "}";
attribute + "state4: {";
attribute + "goto state5 ;\n";
attribute + "}";
attribute + "state5: {";
attribute + "goto state6 ;\n";
attribute + "}";
attribute + "state6: {";
attribute + out1->name() + " = " + component_name + "_R ;\n";
attribute + "goto NEXTSTATE ;\n";
attribute + "}";
}

```

Figure 2e

////////////////////////////////////

// Declare a new functionality //

// which accumulates data //

////////////////////////////////////

FUNCTIONALITYDEF ACCUMULATE {

INPUT a,over;

OUTPUT q;

ADD adder;

DCONNECT(a,adder->in1);

DCONNECT(adder->out1,adder->in2);

}

////////////////////////////////////

// Declaration of a accumulator with a //

// variable latency ///

////////////////////////////////////

RESOURCEDEF ACCUMULATOR\_VAR\_LATENCY

{

////////////////////////////////////

// An Accumulator //

////////////////////////////////////

FUNCTIONALITY ACCUMULATE;

Figure 3a

////////////////////////////////////

// The adder latency is variable. In //

// that case, this marks the number //

// of states in the interface code //

////////////////////////////////////

ATTRIBUTE NUM\_STATES

```
{
    attribute + "2";
}
```

////////////////////////////////////

// Interface access mechanism wherein //

// we have variable latency //

////////////////////////////////////

ATTRIBUTE INTERFACE

```
{
    attribute + "state1: {";
    attribute + "if(" + over->name() + " = '1'){\\n" +
        "goto NEXTSTATE;}\\n";
    attribute + "else { " +
        "goto state2;}\\n";
    attribute + "}";
    attribute + "state2: {";
    attribute + q->name() + "=" + q->name() + "+" + a->name() + ";";
    attribute + "}\\n";
}
}
```

Figure 3b



////////////////////////////////////

// Declare a new functionality //

// which accumulates N data //

////////////////////////////////////

FUNCTIONALITYDEF ACCUMULATE {

INPUT a,N;

OUTPUT q;

ADD adder;

DCONNECT(a,adder->in1);

DCONNECT(adder->out1,adder->in2);

}

////////////////////////////////////

// Declaration of a accumulator with a //

// variable latency ///

////////////////////////////////////

RESOURCEDEF ACCUMULATOR\_VAR\_LATENCY

{

////////////////////////////////////

// An Accumulator //

////////////////////////////////////

FUNCTIONALITY ACCUMULATE;

Figure 4a

```

////////////////////////////////////
// The adder latency is variable and //
// is equal to N where N is an input //
// port of the ACCUMULATE function //
////////////////////////////////////

```

#### ATTRIBUTE NUM\_STATES

```

{
    attribute + "1";
}

```

```

////////////////////////////////////
// Interface access mechanism wherein //
// we have variable latency          //
////////////////////////////////////

```

#### ATTRIBUTE INTERFACE

```

{
    attribute + "state1: {";
    attribute + "for(i = 0;i < " +
        N->name() + ";i = i + 1){";
    attribute + q->name() + "=" + q->name() +
        " + " + a->name() + "};\n";
    attribute + "goto NEXTSTATE;\n";
    attribute + "}\n";
}

```

Figure 4b

$$a = x * y + z$$

(i)

$$a = z + x * y$$

(ii)

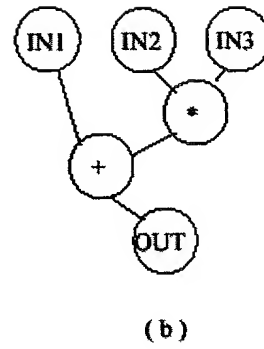
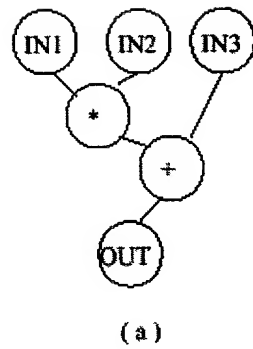


Figure 5